*JIKO (JURNAL INFORMATIKA DAN KOMPUTER)*
Februari 2025, Volume: **9**, No. **1** | Pages 204–211
doi: 10.26798/jiko.v9i1.1576
e-ISSN : 2477-3964 – p-ISSN : 2477-4413

ARTICLE

# Pengembangan Sistem Backup berbasis Backup Flow Microservice Dan GRPC Menggunakan Cloud Computing

## *Backup Flow Microservice And gRPC Based Backup System Development Using Cloud Computing*

Wagito* dan Adnan Nuryono

Informatika, Fakultas Teknologi Informasi, Universitas Teknologi Digital Indonesia, Yogyakarta, Indonesia
*Penulis Korespondensi: wagito@utdi.ac.id

(Disubmit 05-12-24; Diterima 02-01-25; Dipublikasikan online pada 05-02-25)

**Abstrak**
Pada era teknologi informasi diperlukan sistem backup yang efisien dan terukur untuk suatu organisasi. Salah satu teknologi yang dapat dipakai adalah sistem backup flow. Sistem backup flow adalah sistem berbasis microservices yang memecah sistem backup tradisional menjadi beberapa komponen independen, memungkinkan skalabilitas dan adaptasi yang lebih baik. Sistem backup ini bertujuan untuk meningkatkan kinerja dan ketahanan sistem menggunakan pendekatan microservices dan protokol komunikasi gRPC. Protokol komunikasi gRPC digunakan untuk transfer data antar layanan dan peningkatan kinerja. Sebagai penyedia layanan sistem digunakan teknologi cloud computing. Hasil penelitian menunjukkan bahwa proses backup paling baik adalah dilakukan secara bertahap atau terjadwal tanpa melibatkan user. Penggunaan protokol gRPC dalam berkomunikasi antar service dapat mengurangi response time.

**Kata kunci:** gRPC; microservice; backup flow

**Abstract**
In the era of information technology, an efficient and measurable backup system is needed for an organization. One technology that can be used is a backup flow system. Flow backup systems are microservices-based systems that break down traditional backup systems into independent components, allowing for better scalability and adaptability. This backup system aims to improve system performance and resilience using a microservices approach and the gRPC communication protocol. The gRPC communication protocol is used for data transfer between services and performance improvement. As a system service provider, cloud computing technology is used. The research results show that the best backup process is carried out in stages or on a scheduled basis without involving the user. Using the gRPC protocol in communicating between services can reduce response time.

**KeyWords:** gRPC; microservice; backup flow

## 1. Pendahuluan

Aktivitas *backup* merupakan kegiatan yang sangat krusial pada dunia industri khususnya dalam bidang teknologi informasi. Banyak cara yang dilakukan untuk melakukan *backup* data, mulai dari hal yang bersifat *offline* maupun *online* melalui simpanan *cloud*. *Backup* data penting dilakukan karena resiko data hilang bisa kapan saja. Itulah mengapa harus selalu melakukan *backup* data secara berkala. Sarana penyimpanan digital seperti *flashdisk*, *harddisk* eksternal, atau penyimpanan lain, mampu menampung data



**How to Cite:** Wagito *et al.*, "Pengembangan Sistem Backup berbasis Backup Flow Microservice Dan GRPC Menggunakan Cloud Computing", *JIKO (JURNAL INFORMATIKA DAN KOMPUTER)*, Volume: **9**, No.**1**, Pages 204–211, Februari 2025, doi: 10.26798/jiko.v9i1.1576.

yang banyak dan relatif efisien. Sistem tinggal dipasang dan dilakukan penyalinan data. Namun, tingkat efisiensi ini tidak dapat bertahan lama mengingat umur penyimpanan digital tergantung pada kecepatan *read* maupun *write*. Sistem penyimpanan ini juga rentan terhadap kerusakan maupun serangan virus yang beresiko terjadinya kerusakan sistem *backup*.

Di era digital yang semakin canggih, perusahaan hendaknya memiliki strategi atau rencana *backup* data yang aman. Khususnya untuk data yang sangat krusial bagi operasional perusahaan. Menurut riset yang berjudul Indonesia *consumer mobile habit and data management survey* ada sekitar 81 persen atau sekitar 900 dari 1.120 responden yang sadar pentingnya *backup* data. Dari riset diketahui 30 persen responden melakukan backup lebih dari satu kali dalam tiga bulan. Sedangkan, 16 persen melakukan *backup* data kurang dari satu kali satu bulan, dan 20 persen hanya sekali dalam satu bulan. Hal ini dapat dilihat betapa kurangnya pengetahuan akan pentingnya *backup* sebuah data yang dilakukan untuk mengantisipasi adanya kerugian yang diciptakan akibat data hilang.

Dalam penelitian dikembangkan aplikasi *backup* yang dapat melakukan *backup* semua jenis file atau konfigurasi yang dapat di-*restore* sesuai dengan versi aplikasi atau program yang dibutuhkan. *Backup flow* bekerja di latar menggunakan *worker* yang otomatis men-*trigger restore* maupun *backup* menurut jadwal. *Backup* bersifat *cloud* maupun lokal dan *user* dapat memastikan bahwa file yang di *upload* valid dan dapat digunakan pada versi yang *user* pakai saat ini. *Backup flow* menjawab permintaan *user* untuk aplikasi *backup* yang ada sistem penjadwalan otomatis. Tujuan penelitian adalah membangun sistem *backup flow* berbasis *microservices* dan gRPC menggunakan basis *cloud computing*. Hasil sistem ini memberikan alternatif sistem *backup* yang memberikan sistem backup secara terjadwal otomatis.

Beberapa penelitian tentang pengembangan sistem backup menggunakan metode *backup flow* berbasis *microservice* dan gRPC pernah dilakukan. Johannes Fernandes Andry menyusun aplikasi backup yang bertujuan mencegah adanya *crash* atau *error* pada aplikasi, sistem ini menggunakan metode SDLC [1].

Sistem *backup* pernah disusun Kadek Surya Mahedy menggunakan metode paradigma *prototyping*. *Prototyping* merupakan sebuah proses yang memungkinkan pengembang untuk membuat model perangkat lunak yang akan di rekayasa. Hasil yang diperoleh adalah bertujuan agar penggunaan *backup* yang mudah dan juga efisien, dan jika terdapat *error* maka aplikasi akan me-*restore* file dari *server* cadangan [2].

Penelitian *backup* pernah dilakukan Andi Rosano dkk. pada bank XYZ, metode penelitian menggunakan Model *Mirroring* Sistem. Lokasi *Disaster Recovery Center* terletak di kota lain, untuk mengantisipasi hal hal seperti kebakaran dan sebagaimana, hasil dari penelitian ini dinyatakan bahwa dalam 30 menit pertama nampak normal, namun setelah 30 menit selanjutnya terjadi fatal yang mengharuskan sistem kembali ke posisi semula [3].
Aplikasi mengenai *microservice* disusun oleh Maksy Sendiang dkk. berfokus pada pengembangan *Machine Learning* menggunakan teknologi *microservice* akan membentuk aplikasi yang memberikan dampak positif pada proses belajar mengajar [4]. Mohammad Harry Khomas dkk. meneliti efek-efek yang didapatkan dari arsitektur *microservice*. Arsitektur ini dapat disimpulkan tidak mengganggu sistem lain pada saat melakukan pengembangan dan perbaikan sistem karena sistem dibuat secara terpisah sehingga dapat memperkecil ruang lingkup dari pengembangan [5].

Beberapa teori diperlukan untuk mengembangkan sistem aplikasi *backup* dan *restore* menggunakan *microservice* dan gRPC. *Backup* adalah proses duplikasi atau menyalin data atau file dari satu perangkat atau media penyimpanan ke media sekunder, seperti *flashdisk*, *hardisk* eksternal, sistem *cloud* atau media lainnya, yang dilakukan secara *offline* maupun *online*. *Backup* memiliki dua tujuan. untuk mengembalikan data apabila data tersebut hilang, baik karena terhapus atau karena rusak (*corrupt*) dan untuk mengembalikan data ke titik tertentu pada masa lalu [6]. *Restore* adalah proses mengembalikan kembali sebuah data atau file ke tempat semula. Jika data terhapus secara tidak sengaja, maka masih dapat mencari file data tersebut dalam *recycle bin* komputer untuk kemudian dikembalikan ke tempat lokasi semula file itu berada, di suatu folder tertentu [7].

Untuk menyusun sistem *backup* diperlukan *framework* gRPC, *microservice* dan bahasa Golang. *Framework* gRPC merupakan salah satu *framework* RPC *open source* yang dikembangkan oleh Google. Pada gRPC aplikasi klien dapat secara langsung memanggil metode pada aplikasi *server* pada mesin yang berbeda

seolah-olah itu adalah objek lokal dan menjadi mudah membuat aplikasi dan layanan pun terdistribusi [8]. *Microservice* merupakan desain arsitektur untuk membuat sebuah aplikasi yang terdiri dari berbagai unit layanan tersendiri tapi tetap saling terhubung. Setiap unit layanan dalam aplikasi tersebut menjalankan fungsi berbeda, namun tetap mendukung satu sama lain [9]. Golang adalah bahasa pemrograman yang diketik dan dikompilasi secara statis dan menghasilkan kode biner mesin. Menariknya, bahasa pemrograman yang satu ini bersifat *open source*. Golang dihimpun dan ditulis menggunakan bahasa pemrograman C maka, tak heran jika banyak orang menganggap Golang adalah bahasa pemrograman C di abad ke-21. Selain itu, tak heran pula jika banyak orang jadi tertarik untuk belajar Golang [10].

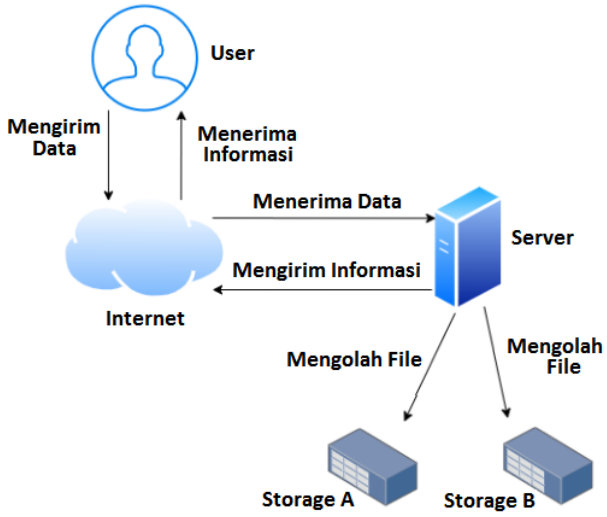
Sistem *backup* diterapkan secara *cloud computing* yang didukung MySQL dan GORM. Menurut *Indonesian Cloud* [11], dinamakan *Cloud Computing* karena informasi yang diakses secara *remote* di "awan" atau ruangan virtual. Perusahaan-perusahaan yang menyediakan layanan *cloud*, memungkinkan para penggunanya menyimpan file dan aplikasi dari *server* jarak jauh. Mereka juga bisa mengaksesnya asalkan ada sambungan internet. Ini berarti, seorang pengguna tidak perlu berada di tempat tertentu untuk mendapatkan akses file-nya. MySQL adalah sebuah *database management system* (manajemen basis data) menggunakan perintah dasar SQL (*Structured Query Language*) yang merupakan suatu bahasa yang dipakai di dalam pengambilan data pada *relational database* atau database yang terstruktur. Jadi MySQL adalah *database management system* yang menggunakan bahasa SQL sebagai bahasa penghubung antara perangkat lunak aplikasi dengan *database server* [12]. GORM adalah sebuah *library* dengan menggunakan bahasa pemrograman Golang untuk mendukung penggunaan SQL (*Structure Query Language*). GORM dapat mendukung aplikasi database seperti MySQL, PostgreSQL dan SQLlite [13].

## 2. Metode

Bahan atau data yang dibutuhkan pada penelitian ini meliputi file yang di *upload* dan informasi berkaitan dengan *user*. Perangkat keras yang dibutuhkan untuk mengembangkan meliputi laptop menggunakan processor Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz (4 CPUs) dan simpanan RAM 8 GB. Sedangkan perangkat lunak yang digunakan untuk menerapkan sistem meliputi Sistem Operasi Windows 10, Sistem Operasi Linux, sistem simpanan MySQL, GORM, KrakenD [14, 15], protokol gRPC [16], bahasa Golang [17, 18], Visual Studio Code, Git dan Virtual Machine.

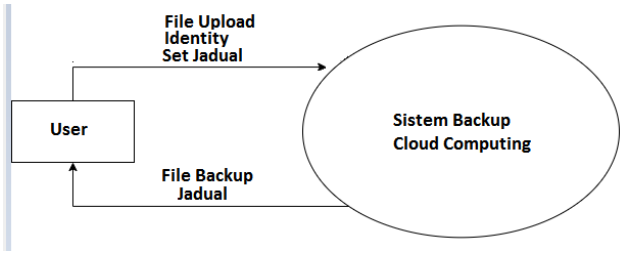
Masukan yang diperlukan untuk menjalankan sistem meliputi informasi *user* sebagai pelaku utama pada sistem, file *secret* pada tiap *cloud computing*, *schedule* untuk menentukan kapan waktunya *backup* kapan waktunya *restore* dan file yang di *upload* pada sistem. Keluaran yang dihasilkan oleh sistem berupa informasi mengenai jadwal *backup*, *restore* dan status keluaran *fail* atau *success* dalam proses *backup* atau *restore*. Keluaran lain dari sistem file hasil dari restore yang mana file tersebut menuju lokasi yang telah ditentukan oleh *user*.

Pengumpulan data dilakukan dengan langkah-langkah studi kepustakaan dan observasi. Studi kepustakaan digunakan untuk pengumpulan data berdasarkan literasi dari buku referensi dan dokumentasi lain yang berkaitan dengan masalah yang sedang diteliti. Observasi digunakan untuk pengumpulan data dengan melakukan pengamatan langsung untuk menguji keberhasilan sistem. Perancangan arsitektur pada pengembangan sistem *backup flow* ditunjukkan pada Gambar 1.




**Gambar 1.** Arsitektur Sistem

Sistem dapat digunakan *user* apabila terkoneksi dengan internet, karena sistem *backup* dan *restore* membutuhkan internet dan kondisi laptop. *User* dapat mengirim data seperti file, mengatur jadwal, konfigurasi *storage*, kemudian *server* akan menerima data tersebut yang nantinya akan disimpan di *storage* A maupun B, lalu *server* akan merespon balik berupa informasi tentang status *backup* maupun *restore*. Secara konteks interaksi *user* dan sistem digambarkan pada Gambar 2.




**Gambar 2.** Diagram Konteks

*User* berperan memiliki peran untuk melakukan *upload* file, melakukan *input* identitas atau kredensial dari *storage*, mengatur jadwal, dan melakukan *restore* objek file yang telah di *upload*.

## 3. Hasil dan Pembahasan

Metode *backup flow* berbasis *microservices* memecah *service* menjadi berbagai bagian dan tiap bagian mengurus kegiatan serta fungsinya masing-masing. Perantara dibutuhkan untuk komunikasi antar *service*. Perantara yang bisa digunakan adalah HTTP atau gRPC. Pada penelitian ini digunakan perantara berupa gRPC. Perantara gRPC dinilai lebih stabil dan lebih cepat.

### 3.1 *User Service*

*User Service* merupakan komponen kunci dalam arsitektur *microservice* yang memiliki tanggung jawab utama dalam manajemen pengguna. Dengan fokus pada proses *login*, registrasi, dan penyediaan informasi terkait pengguna, *User Service* menjadi fondasi bagi interaksi pengguna dalam sistem. Saat pengguna mendaftar, layanan ini memvalidasi dan menyimpan informasi ke dalam penyimpanan data yang sesuai. Ketika *login*, *User Service* memastikan keamanan dan otentikasi, menghasilkan *token* yang memungkinkan akses yang sah ke layanan lain. Selain itu, *User Service* memberdayakan pengguna untuk mengelola profil mereka, termasuk manajemen kata sandi, pemulihan akun, dan melihat informasi terkait akun. Dengan fokus pada keamanan dan kenyamanan pengguna, *User Service* menjadi pilar fundamental dalam membangun pengalaman pengguna yang terpercaya dan efisien dalam lingkungan *Microservice*.

Ekspektasi yang akan diperoleh dari adanya *user service* ini adalah keberhasilan dalam mengolah data *user* ditampilkan dalam Tabel 1.

**Tabel 1.** Hasil Uji *User Service*

| Fitur | Hasil | *Response Time* |
|---|---|---|
| *Login* | sukses | 250 detik |
| Register | sukses | 1.6 detik |
| *Profile* | sukses | 450 mili detik |

Dari data hasil pengamatan dapat dilihat bahwa waktu respons sistem dalam menanggapi permintaan dipandang cukup cepat. Namun demikian, pada prosedur *login* nilainya masih dipandang cukup besar, sehingga perlu perbaikan pada proses ini.

### 3.2 *Storage Service*

*Storage Service* adalah sebuah *boilerplate* yang sama hal nya seperti *User Service*. Menerima *request* HTTP dan mengirim ke gRPC untuk proses lebih yang lanjut. Namun ada suatu pembeda pada *storage-service* yaitu menghubungkan akses *storage* ke *server*. Untuk implementasi pada *storage-service* ambil saja contoh pada kasus *dropbox*, di mana *dropbox* membutuhkan *authorization*. Sedangkan *firebase* hanya membutuhkan *credential* yang di-*generate* tanpa harus mengakses *link authorization oauth*. Setelah mendapatkan *respons*, lanjut untuk menyimpan data respon tersebut pada database, untuk nantinya dipakai dalam bertransaksi dengan *Dropbox*.

Ekspektasi yang akan diperoleh dari adanya *storage service* ini adalah keberhasilan dalam mengolah data *user* seperti dalam Tabel 2.

**Tabel 2.** Hasil Uji *Storage Service*

| Fitur | Hasil | *Response Time* |
|---|---|---|
| Verifikasi *Firebase* | sukses | 450 mili detik |
| Verifikasi *Dropbox* | sukses | 450 mili detik |
| *List Storage* | sukses | 390 mili detik |
| *Upload* File < 100kb | sukses | 1.71 detik |
| *Upload* File < 20 mb | sukses | 3.50 detik |
| *Download* File < 100 kb | sukses | 700 mili detik |
| *Download* File < 20 mb | sukses | 2.10 detik |

Dapat dilihat pada data bahwa proses verifikasi dan proses *listing* file pada *server* dirasa cukup layak, sedangkan untuk proses *upload* dan *download*, angkanya sebanding dengan ukuran file.

### 3.3 *Schedule Service*

*Schedule service* berguna untuk mengatur penjadwalan terhadap *backup* maupun *restore* file-file yang di *upload* oleh *user* ke tiap-tiap *storage*, *schedule service* ini menyimpan data-data penjadwalan dari *user* yang nantinya dari *agent* akan mengambil data tersebut dan mencocokkan tiap waktu apakah waktu yang didapatkan harus melakukan *backup* atau *restore*. Ekspektasi yang akan diperoleh dari adanya *schedule service* ini adalah keberhasilan dalam mengolah data *user* dirinci dalam Tabel 3.

**Tabel 3.** Hasil Uji *Schedule Service*

| Fitur | Hasil | *Response Time* |
|---|---|---|
| *Add Schedule* | sukses | 395 mili detik |
| *List Schedule* | sukses | 395 mili detik |
| *Detail Schedule* | sukses | 397 mili detik |
| Delete Schedule | sukses | 406 mili detik |

Dapat dilihat pada data bahwa proses-proses *schedule* berjalan dengan sukses dengan waktu yang cukup cepat. *Response time* yang dihasilkan nilainya pada orde ratusan mili detik.

### 3.4 *Notification Worker Service*

*Notification Worker* berfungsi sebagai pengirim notifikasi, notifikasi yang saat ini berjalan adalah notifikasi dari email. *Notification* ini menggunakan sistem *queue* menggunakan rabbitMQ. Kenapa rabbitMQ, selain karena gratis dan juga bersifat *self hosted*, rabbitMQ ini memiliki sistem *queue* yang baik selain untuk *publish* dan *subscribe*. rabbitMQ ini digunakan karena ada sistem yang namanya *requeue* dan *rejected*, *requeue* ini digunakan untuk melakukan antrian kembali yang bertujuan untuk mengirimkan email jikalau terdapat suatu hal *error* diluar *service down*, misal saat mengirim email terkena *timeout*, maka dari *server* akan melakukan *requeue*. Sedangkan *rejected* digunakan untuk menolak *queue* tersebut diolah kembali atau di antri kembali. Kasus untuk *rejected* ini bervariasi dari *error* yang terjadi oleh *server*, email *publisher* maupun dari *database error*. Ekspektasi yang akan diperoleh dari adanya *notification service* ini adalah keberhasilan dalam mengolah data *user* seperti tertulis dalam Tabel 4.

**Tabel 4.** Hasil Uji *Notification Service*

| Fitur | Hasil |
|---|---|
| Verifikasi Email | sukses |
| Menerima Email | Sukses menerima email ke *providers* |
| Proses kirim email | Sukses kirim email ke alamat tujuan |

Dari tabel dapat dilihat bahwa sistem berhasil melakukan semua proses yang berkaitan dengan notifikasi secara benar. Pada bagian ini, hanya dilakukan pengujian berjalannya sistem.

### 3.5 *Agent Worker Service*

*Agent Worker* berfungsi sebagai *executor* dari adanya *schedule* yang diberikan oleh *user*, *agent worker* ini berjalan selama satu detik dalam waktu 24 jam, bisa dikatakan bahwa *agent worker* ini bersifat *real time worker*, yang artinya *worker* yang dijalankan atau mengambil data secara *realtime* tanpa adanya jeda. *Agent worker* ini berjalan pada *host user*, yang artinya *user* diharuskan untuk menjalankan *worker* di *device* milik mereka.

*Agent worker* punya dua tugas utama, yaitu bekerja saat ada jadwal *backup* dan bekerja saat ada jadwal *restore*. Fungsi *backup* pada *agent worker* ini adalah *backup*, karena tugasnya adalah untuk memastikan bahwa file yang di *upload user* dan dijadwalkan dalam keadaan *up-to-date*. Yang berarti filenya yang di-*upload* dalam keadaan terbaru.

**Tabel 5.** Hasil Uji *Backup Agent*

| Fitur | Hasil | *Response Time* |
|---|---|---|
| *Backup* 52 kb | sukses | waktu singkat |
| *Backup* 2.6 mb | sukses | 6 detik |
| *Backup* 13 mb | sukses | 37 detik |

Fungsi *restore* pada *agent worker* ini adalah *restore*, karena tugasnya adalah untuk memastikan bahwa file yang berada di *device user* dan dijadwalkan dalam keadaan *up-to-date*. Yang berarti filenya yang di-*download* dalam keadaan terbaru. Mekanismenya sama dengan *backup* namun yang diperhatikan adalah *restore* ini akan menaruh filenya berada di *folder* yang *user* kehendaki, jika *folder* tersebut tidak ada maka status tersebut akan menjadi *failed.*

## 4. Simpulan

Berdasarkan hasil analisis dan pembahasan disimpulkan bahwa sistem *backup* menggunakan protokol gRPC berbasis *microservices* berhasil dibangun serta berjalan secara baik. Pada hasil ditemukan bahwa cara paling efisien dalam melakukan *backup* adalah secara bertahap atau terjadwal tanpa melibatkan *user* dalam prosesnya. Penggunaan protokol gRPC dalam komunikasi antar *service* dapat mengurangi *response time.*

## Pustaka

[1] J. F. Andry, "Pengembangan aplikasi backup dan restore secara otomatis menggunakan sdlc untuk mencegah bencana," 2017.

[2] K. S. Mahedy, "Implementasi sistem backup data pada sistem informasi perpustakaan universitas pendidikan ganesha," 2021.

[3] A. Rosano and D. Sudaradja, "Manajemen backup data untuk penyelamatan data nasabah pada sistem informasi perbankan (studi kasus: Pt bank xyz)," 2020.

[4] M. Sendiang, S. Kasenda, and J. Purnama, "Implementasi teknologi microservice pada pengembangan mobile learning," *Journal of Applied Informatics and Computing*, 2018.

[5] M. H. K. Saputra and L. M. Nabil, "Penerapan arsitektur microservice pada sistem tata kelola mata kuliah proyek politeknik pos indonesia," *Jurnal Teknik Informatika*, 2021.

[6] Nagitect. (2022) Fungsi dan manfaat backup bagi perusahaan. [Online]. Available: https://nagitec.com/fungsi-dan-manfaat-backup-bagi-perusahaan/

[7] R. Pebriasari. (2022) Backup, restore, dan recovery data. [Online]. Available: https://blog.wowrack.co.id/2017/11/backup-restore-dan-recovery-data.html

[8] H. Mustafa. (2019) Belajar membuat grpc microservice dengan line armeria dan spring webflux. [Online]. Available: https://www.dicoding.com/blog/belajar-membuat-grpc-microservice-dengan-line-armeria-dan-spring-webflux/

[9] Benefita. (2022) Microservices: Pengertian, contoh, dan kelebihannya. [Online]. Available: https://www.niagahoster.co.id/blog/microservices/#Apa_Itu_Microservices

[10] V. Osadchiy. (2022) Why use the go language for your project? [Online]. Available: https://yalantis.com/blog/why-use-go/

[11] Indonesiancloud. (2022) Mengenal cloud computing: Pengertian, tipe, dan fungsinya. [Online]. Available: https://indonesiancloud.com/mengenal-cloud-computing/

[12] Y. K. (2022) Pengertian mysql, fungsi, dan cara kerjanya (lengkap). [Online]. Available: https://www.niagahoster.co.id/blog/mysql-adalah/?amp

[13] A. Khair. (2020) Back-end basic with go part 1: Melakukan crud menggunakan gin-gonic dan gorm. [Online]. Available: https://medium.com/back-end-basic-with-go/simple-e-commerce-part-1-melakukan-crud-menggunakan-gin-onic-dan-gorm-d1b4fb6ff548

[14] krakend.io. (2022) Introduction to krakend. [Online]. Available: https://www.krakend.io/docs/overview/

[15] N. S. Gill. (2021) Understanding krakend api gateway for microservices. [Online]. Available: https://www.xenonstack.com/insights/krakend-for-microservices

[16] GRPC.io. (2022) Basics tutorial. [Online]. Available: https://gRPC.io/docs/languages/go/basics/

[17] gorm.io. (2022) The fantastic orm library for golang. [Online]. Available: https://gorm.io/docs/

[18] G. N. Arviana. (2020) Belajar bahasa pemrograman "golang" untuk atasi traffic tinggi. [Online]. Available: https://glints.com/id/lowongan/belajar-golang/#.Y4ly_HZBxhF